**String类特点分析**

1. String 比较

“ == ”：进行的数值上的比较，当用于对象时比较的是两个内存地址的数值比较 ;

“equals():”是提供的比较方法，可以直接判断字符串内容 ;

1. 字符串常量是String类的匿名对象
2. 任何使用 "" 定义的String 类都是一个String类的匿名对象

eg:

String str = "Str";

str是一个引用名字；

关于对象相等判断的小技巧: 在以后的开发中，如果某些时候数据是由用户输入，并且要求这些数据为一个指定内容的情况下建议将字符串的一个常量写在前面； equals()方法里面有一个自动回避空的判断，那么如果将字符串变量放前面则可以自动避免用户输入为空的情况。

1. String 类的两种实例化的区别；
2. .分析直接赋值的对象实例化模式

在程序中只需要将一个字符串赋值给String 类对象就可以实现对象的实例化处理

String str = "mld";

这种情况下肯定是只会开辟一块堆内存空间;除了这种内存模式以外，利用直接赋值实例化String 对象还可以实现同一个字符串对象数据的共享操作

范例:

String strA = "mld";

String strB = "mld";

System.out.println(strA == strB); //比较的是两个引用名称的地址空间,此时strB 是不会开辟新的内存空间 也就是说 strA == strB 为true；出现这样的原因主要是因为在Java程序底层中提供一个专门的字符串池（字符串数组）当我们想要创建 strB 所指向的字符串内容 的时候会先到字符串池 中进行查找,如果有,会直接指向字符串池中的地址，所strA 与 strB 所指向的地址相同 因此strB == strA

1. 分析构造方法实例化

构造方法进行对象实例化可以说是在进行对象定义时的一种常见方法，因此Java中也提供String 类的构造方法实例化的方法

String str = new String( "mld" );

此时 "mld" 匿名对象会创建一个堆空间 ，而new 也会创建一个堆空间 ，str 指向的便时new 关键字创建的空间，由此可见 使用 new 构造方法进行对象的实例化会导致 "mld" 匿名对象创建一个垃圾空间（浪费空间）

范例: 更换一种形式

String strA = "mld" ;

String strB = new String( "mld" );

这样在 strB 中的 匿名对象创建堆空间时 会先在Java字符串池中进行查找。

而构造方法 new 关键实例化对象时不会出现自动保存到字符串池中的处理。

String strA = "mld" ;

String strB = new String( "mld" );

System.out.println(strA == strB); //false;

可以发现构造方法实例化对象时会开辟一个专属的空间，而不会保存到字符串池中，但是Java中有帮助开发者手动入池的方法: public String intern() ;

eg:

String strA = "mld" ;

String strB = new String( "mld" ).intern();

System.out.println(strA == strB); //true;

**·请解释String 类两种对象实例化方式的区别？**

（1）直接赋值: 只会产生一个实例化对象，并且可以自动保存到字符串池当中，以实现该字符串实例的重用

（1）构造方法：会产生两个实例化对象，并且不会自动入池，无法实现对象重用，但是可以使用intern（）方法进行手动入池

1. String对象（常量）池

对象池的主要目地时实现数据共享处理。以String 对象池为例，里面的内容主要是为了重用，而重用实际上就属于共享设计，但是在Java中对象（常量）池实际上分为两种:

·静态常量池：指的时程序（\*.class） 在加载的时候会自动将此程序中保存的字符串、普通的常量、类和方法的信息等全部进行分配；

·运行时常量池: 当一个程序（\*.class） 加载之后，里面可能会有一些变量，这个时候提供的一个常量池我们管他叫

运行时常量池；

|  |
| --- |
| 范例:（静态常量池）  public class StringDemo {  public static void main(String args[]){  String strA = "www.mld.cn";  String strB = "www." + "mld" +".cn";  System.out.println(strA == strB); //true  }  } |

本程序中所给出的内容全部都是静态常量数据（字符串的常量都是匿名对象），所以最终在程序加载的时候会自动帮助开发者处理好相应的连接；

|  |
| --- |
| 范例:(运行时常量池)  public class StringDemo {  public static void main(String args[]){  String info = "mld";  String strA = "www.mld.cn";  String strB = "www." + info +".cn";  System.out.println(strA == strB); //false;  }  } |

这个时候之所以是一个false ， 是因为程序在加载的时候不确定info 是什么内容。因为我们在字符串连接的时候info 采用的是一个变量。那么变量的内容是可以修改的 ，所以他不认为最终strB 的最终结果就是一个所需要的最终的结果。所以 strB 存在运行时常量池中 因此在这个范例中strB != strA

1. 字符串内容不可修改

1. 在String类之中包含的是一个数组，而数组最大的一个缺点就是长度不可改变。也就是说我们设置了一个字符串之后，会自动进行一个数组空间的开辟，开辟的内容长度是固定的。

|  |
| --- |
| 范例:  public class StringDemo{  public static void main(String args[]){  String str = "www.";  str += "mld";  str += ".cn";  System.out.println(str);  }  } |

分析以上案例：

首先我们创建了一个堆空间 "www." ,而在 str += "mld" 时 会先创建一个 "mld" 的新空间，但新空间并不是我们所需要的，我们需要的是"www.mld" 因此会再创建一个 "www.mld" 的新空间，此时str 指向改变，指向了新的空间 "www.mld" ,而后面的 str += ".cn"; 和前面的也是一样的;

通过上面的分析我们可以知道，字符串常量的内容并没有发生任何改变，改变的只是一个String类对象的引用，并且这样的引用会带来大量的垃圾空间;

分析下面的程序:

|  |
| --- |
| public class StringDemo{  public static void main(String args[]){  String str ="www.";  for(int x = 0; x < 1000 ; x ++){  str += x;  }  }  } |

像这样的程序如果真的执行的,那么将会产生 1000 多个垃圾空间，并且String 对象的指向会修改 1000多次，所以再以后的开发中， 我们应该尽量避免修改字符串

1. Java中的主方法
2. Java中的主方法组成是非常复杂的，并且单词也很多: public static void main(String args[])

·public : 描述的是一种访问权限，主方法是一切的开始点，一定是公共的；

·static : 程序的执行是通过类名称完成的，所以表示此方法是由类直接调用;

·void : 主方法是一切的起点，一旦开始就没有返回的可能，所以是void；

· main : 是一个系统定义好的方法名称；

·String args[] : 字符串的数组，可以实现程序启动参数的接收。

范例:(输出启动参数)

public class StringDemo{

public static void main(String args[]){

for(String arg : args){

system.out.println(arg);

}

}

}

在程序执行的时候可以设置参数，每一个参数之间使用空格分隔;

javac StringDemo.java

java StringDemo first second

此时就会输出

first

second

java StringDemo "first second" "hello world"

此时就会输出

first second

hello world

以后可以通过这种启动参数实现数据输入的的模拟。